







# *KluSIM*: Speeding up *K*-Medoids Clustering over Dimensional Data with Metric Access Method

Larissa R. Teixeira<sup>1</sup><sup>a</sup>, Igor A. R. Eleutério<sup>1</sup><sup>b</sup>, Mirela T. Cazzolato<sup>1,2</sup><sup>c</sup>, Marco A. Gutierrez<sup>2</sup><sup>d</sup>,  
Agma J. M. Traina<sup>1</sup><sup>e</sup> and Caetano Traina-Jr.<sup>1</sup><sup>f</sup>

<sup>1</sup>*Institute of Mathematics and Computer Science (ICMC), University of São Paulo (USP), São Carlos, Brazil*

<sup>2</sup>*The Heart Institute (InCor), University of São Paulo (USP), São Paulo, Brazil*

**Keywords:** Dimensional Data, *k*-medoids, Clustering, Indexing, Metric Access Method.

**Abstract:** Clustering algorithms are powerful data mining techniques, responsible for identifying patterns and extracting information from datasets. Scalable algorithms have become crucial to enable data mining techniques on large datasets. In literature, *k*-medoid-based clustering algorithms stand out as one of the most used approaches. However, these methods face scalability challenges when applied to massive datasets and high dimensional vector spaces, mainly due to the high computational cost in the swap step. In this paper, we propose the *KluSIM* method to improve the computational efficiency of the swap step in the *k*-medoids clustering process. *KluSIM* leverages Metric Access Methods (MAMs) to prune the search space, speeding up the swap step. Additionally, *KluSIM* eliminates the need of maintaining a distance matrix in memory, successfully overcoming memory limitations in existing methodologies. Experiments over real and synthetic data show that *KluSIM* outperforms the baseline *FasterPAM*, with a speed up of up to 881 times, requiring up to 3,500 times fewer distance calculations, and maintaining a comparable clustering quality. *KluSIM* is well-suited for big data analysis, being effective and scalable for clustering large datasets.


## 1 INTRODUCTION


Clustering is a data mining task applied across different applications, such as bio-informatics, image processing, pattern recognition, financial risk analysis, and more (Qaddoura et al., 2020). Clustering is the process of identifying patterns in a dataset through grouping into clusters. Objects within the same cluster have higher similarity to each other compared to objects in different clusters. Among several clustering algorithms in the literature, the most popular are *k*-means and *k*-medoids, whereas the most employed variant of the latter is the Partitioning Around Medoids (*PAM*) (Kaufman, 1990). The *k*-means algorithm is based on defining a centroid, which is the arithmetic mean of all objects within a cluster. On the


other hand, *k*-medoids are algorithms that seek for a medoid, an actual object within a cluster that have the smallest distance sum to all other objects.


One of the advantages of *k*-medoids over *k*-means algorithm is the results interpretability (Kenger et al., 2023). By assigning actual objects as cluster representatives, *k*-medoids results can provide insights into the dataset based on the characteristics of the medoids.


Medoid-based approaches have shown high clustering quality. However, these methods show many difficulties when running over large amounts of data and high dimensional vector spaces. The *k*-medoid-based algorithms rely on accurate initialization heuristics to preselect initial medoids as seed before the *swap* step. Good seeds can accelerate the convergence of the algorithm and avoid distance calculations during the *swap* step. The *swap* step iterates over the entire dataset to find the best combination of non-medoid and medoid that minimizes a objective function. This step is **computationally costly** and is the bottleneck of the algorithm. Several approaches maintain a in-memory distance matrix to speed up


<sup>a</sup> <https://orcid.org/0009-0007-9917-4404>

<sup>b</sup> <https://orcid.org/0009-0007-3987-8880>

<sup>c</sup> <https://orcid.org/0000-0002-4364-010X>

<sup>d</sup> <https://orcid.org/0000-0003-0964-6222>

<sup>e</sup> <https://orcid.org/0000-0003-4929-7258>

<sup>f</sup> <https://orcid.org/0000-0002-6625-6047>

*swap* and avoid recomputing the distance between the same objects. However, this option is unfeasible for large datasets, due to limited memory.

We focus on clustering dimensional data over large datasets. We propose *KluSIM*, a method that speeds up the *swap* step of medoid-based approaches by taking advantage of Metric Access Methods (MAMs) to index and prune the search space, without trading the semantic meaning of generated clusters for performance. Thus, *KluSIM* avoids the need for maintaining a distance matrix and reduces the number of distance calculations. The key contributions of our work are:

- **MAM-Based Space Pruning.** We strategically take advantage of MAMs for search space pruning during *swap*, drastically reducing the number of distance calculations. We also eliminate the need for precomputing and keeping a in-memory distance matrix. The MAM allows performing a range query over the candidates of medoid objects, even over large high-dimensional datasets. The range query operation associates each non-medoid object to the cluster with its nearest medoid. *KluSIM* computes the minimum distance, for each medoid, among other medoids, and uses it to prune the search space.
- **Centroid-Guided Candidate Selection.** *KluSIM* computes the centroid of each cluster and selects its nearest objects through a  $k$ -NN query to the centroid. The answers are the candidates to be tested as medoid objects during *swap*. Thus, unlike existing methods, *KluSIM* reduces the medoids candidates to be evaluated at each iteration, also reducing the number of distance calculations needed in the overall clustering process.
- ***KluSIM* is Fast and Effective.** Our method is up to 881 faster than *FasterPAM*, the medoid-based clustering approach baseline, while maintaining comparable cluster quality.

*KluSIM* combines both  $k$ -NN and range queries during clustering. Our experimental evaluation shows a significant improvement in the execution time, with comparable quality. *KluSIM* improves efficiency, captures spatial relationships, and enhances the overall clustering process.

**Paper Outline.** Section 2 gives the background. Section 3 reviews the related work. Section 4 introduces the proposed method *KluSIM*. Section 5 presents the experimental evaluation and discussion. Finally, Section 6 concludes this work.

## 2 BACKGROUND

We present the relevant background on clustering algorithms,  $k$ -medoids approaches, and metric access methods. Table 1 shows the symbols and acronyms.

### 2.1 Clustering Algorithms

Clustering, as a data mining technique, partitions a dataset into  $k$  groups, where objects within the same group have high similarity among themselves, while maintaining dissimilarity with objects from other groups (Han et al., 2011). The literature presents several clustering algorithms, mainly categorized into hierarchical and partitioning methods, sometimes also including density-based approaches (Ran et al., 2023). Hierarchical clustering builds a cluster tree structure, also called dendrogram. The objects are organized into a hierarchy of clusters by iteratively merging (agglomerative approach) or dividing (divisive approach) them based on a distance measure (Ran et al., 2023). Despite the dendrograms being effective for small datasets, they do not provide a flat partitioning of the data. Therefore, simpler alternatives are generally preferred by users, such as partitioning methods (Schubert and Rousseeuw, 2021).

Partitioning methods assign each object to one of a predefined number of clusters (Han et al., 2011). The goal is to optimize a certain objective function, such as minimizing the within-cluster variance or maximizing inter-cluster distances. Examples include  $k$ -means and  $k$ -medoids (Kaufman, 1990). Given the better results interpretability of clusters achieved by  $k$ -medoid-based algorithms compared to  $k$ -means, this paper focuses on the former.

#### 2.1.1 $k$ -medoids Algorithms

The Partitioning Around Medoids (*PAM*) (Kaufman, 1990) was one of the first  $k$ -medoid-based algorithms introduced in the literature. *PAM* has two main steps:

- ***build*:** Select  $k$  objects as initial medoids with a greedy search.
- ***swap*:** Based on the initial medoids, *PAM* repeatedly performs the *swap* step to improve the clustering quality until convergence. At each iteration, a non-medoid object is evaluated to improve the clustering quality by replacing one of the current medoids with the non-medoid object.

#### 2.1.2 Cluster Quality Measure

As part of the cluster quality assessment, many techniques have been proposed in the literature. The clus-

Table 1: Summary of symbols and definitions.

Symbol	Description
$k$	Number of clusters
$n$	Number of objects in the dataset
$S$	Set of objects
$s_i, s_j, s_k, s_q$	Objects $\in S$
$\delta$	Dissimilarity (distance) measure
$M$	Set of medoids
$m_i$	A medoid object $\in M$
$\mathcal{C}$	Set of $k$ clusters
$C$	A cluster
$x_C$	An object $\in C$
$p$	Number of nearest neighbors
$S_p$	Set of objects returned from $kNNq$
$TD$	The total deviation of cluster quality
$SS$	Silhouette score
$DBI$	Davies Bouldin Index
$\xi$	Range query radius

tering quality of  $k$ -medoids is evaluated using the absolute error criterion denoted as total deviation ( $TD$ ), defined as:

$$TD = \sum_{i=1}^k \sum_{x_c \in C_i} \delta(x_c, m_i) \quad (1)$$

which is the sum of distances from each object  $x_c \in C_i$  to the medoid  $m_i$  of its cluster (Schubert and Rousseeuw, 2021).

The Silhouette Score ( $SS$ ) is another popular measure to evaluate clustering quality (Lenssen and Schubert, 2024). For each object, the Silhouette Score considers two key metrics: i. The average distance of a object  $i$  to other objects in the same cluster. ii. The average distance of a object  $i$  to other objects in the nearest neighbor cluster. The overall  $SS$  is the average of individual  $SS$  across all objects. The score ranges from  $-1$  to  $1$ , where a higher score means better-defined and well-separated clusters.

Another cluster quality metric is the Davies-Bouldin Index ( $DBI$ ) (Davies and Bouldin, 1979). The index considers both intra-cluster similarity and inter-cluster dissimilarity to calculate the average similarity of each cluster with all others. The  $DBI$  index is obtained as the sum of average similarity values of all clusters, divided by the number of clusters. Lower  $DBI$  indicates a better clustering result (minimum index is zero).

## 2.2 Metric Access Methods

Metric Access Methods (MAMs) are built on top of indexing structures designed to optimize the search for nearest neighbors within a dataset. These methods take advantage of the inherent properties of met-

ric spaces, such as the triangular inequality, to prune subsets of the search space, thus speeding up query processing. Examples of MAMs include the VP-Tree (Yianilos, 1993), Ball-Tree (Omohundro, 1989) and KD-Tree (Bentley, 1975).

A metric space is represented as a domain of objects and a distance function  $\delta$  between two objects  $s_i$  and  $s_j$  from the domain that satisfy the following **properties of a metric space** (Zezula et al., 2006): i. **non-negativity**:  $s_i \neq s_j \rightarrow \delta(s_i, s_j) > 0$ ; ii. **identity**:  $\delta(s_i, s_j) = 0 \Leftrightarrow s_i = s_j$ ; iii. **symmetry**:  $\delta(s_i, s_j) = \delta(s_j, s_i)$ ; iv. **triangle inequality**:  $\delta(s_i, s_k) \leq \delta(s_i, s_j) + \delta(s_j, s_k)$ .

There are two main elementary similarity queries in a metric space (Zezula et al., 2006):

- i. **Range Query ( $Rq$ )**: given a query object  $s_q$ , a distance function  $\delta$  and a radius  $\xi$ , the operation retrieves all objects that differ from  $s_q$  by at most the distance  $\xi$ .
- ii. **k-Nearest Neighbors ( $kNNq$ )**: retrieves the  $k$  objects closest to the query object  $s_q \in S$  considering the distance function  $\delta$ .

## 2.3 Vantage Point Tree

One notable MAM is the Vantage Point Tree ( $VP$ -Tree) introduced by (Yianilos, 1993). Using a ball partitioning approach,  $VP$ -Tree selects a vantage point ( $vp$ ) to partition a dataset  $S$  into two subsets  $S_1 \subset S$  and  $S_2 \subset S$ . This partitioning is based on the median distance  $\bar{x}$  between the chosen  $vp$  to all other objects. The objects are distributed to  $S_1$  or  $S_2$  as:  $S_1 \leftarrow \{s_i | \delta(vp, s_i) \leq \bar{x}\}$  and  $S_2 \leftarrow \{s_i | \delta(vp, s_i) > \bar{x}\}$  where  $vp \in S$ . After this, the same process is recursively applied to each subset, creating a hierarchy.

## 3 RELATED WORK

In this work, we focus on improving the efficiency of  $k$ -medoids. Table 2 summarizes related studies. Here we compare our proposal *KluSIM* with related methods considering the following aspects:

- **Memory Optimization**: most algorithms require precomputing a distance matrix, which is a bottleneck for large datasets. This aspect informs whether the algorithm can perform without an in-memory matrix, showing a significant advantage for the specific approach.
- **Large Datasets**: this aspect informs whether the algorithm runs over the entire dataset within a feasible time frame.

Table 2: *KluSIM* optimizes the usage of memory when performing clustering through a space-pruning technique. Comparison of state-of-art algorithms with our proposed.

Works	Memory optimization	Executes with whole dataset	Main memory	Space-pruning SWAP optimization
PAM-SLIM	✓	✗	✗	✗
SFKM	✗	✓	✓	✗
FastPAM1	✗	✓	✓	✗
FasterPAM	✗	✓	✓	✗
<b>(Proposal) <i>KluSIM</i></b>	✓	✓	✓	✓

- **Main Memory:** this aspect informs whether the algorithm runs in main memory, which is faster than algorithms operating in secondary memory.
- **Space-Pruning swap Optimization:** whether the algorithm takes advantage of space-pruning heuristics, which can significantly enhance the efficiency of the *swap* step in *k*-medoids algorithms by avoiding many distance calculations.

The discussion of the related works is organized into two parts, each addressing key aspects of enhancing the efficiency of *k*-medoid algorithms. Subsection 3.1 explores alternative approaches for *k*-medoids initialization. Subsection 3.2 discusses techniques related to *k*-medoids *swap* optimization.

This organization provides a comprehensive comparison of the proposed *KluSIM* method with existing methodologies.

### 3.1 *k*-medoids Initialization Methods

*PAM*'s *build* step is well known for being state-of-the-art heuristic for initializing *k*-medoids. However, alternative approaches for initialization have been explored in the literature, such as the random selection of initial medoids (Schubert and Rousseeuw, 2021).

In (Arthur and Vassilvitskii, 2007) the authors propose a seeding strategy to improve the *k*-means initialization, denoted as *k*-means++. The algorithm randomly selects the first centroid from the dataset. Then, *k*-means++ selects the next centroids with the probability proportional to their squared distance to the nearest centroid. New centroid are likely to be far from the previously chosen ones.

In (Barioni et al., 2008) the authors present a variation of *PAM* called PAM-SLIM. The algorithm uses the Slim-Tree MAM (Traina et al., 2002), stored in secondary memory, to assist the selection of initial medoids. The algorithm leverages the tree structure to choose the initial medoids, selecting objects located at the middle level of the tree as possible medoids. Unfortunately, as this method does not consider the

entire dataset when selecting the initial medoids, it may affect the clustering quality. Additionally, PAM-SLIM does not take advantage of the benefits of MAM to optimize the *k*-medoids *swap* step. It only leverages the advantage of how the tree is organized, focusing to select a subset of the dataset rather than considering the whole dataset.

In (Park and Jun, 2009) the authors propose the Simple and Fast *k*-medoids (SFKM), an initialization method for *k*-medoids that operates like *k*-means. SFKM computes a distance matrix between all objects, aiming at finding new medoids in each iteration. The algorithm selects the *k*-medoids with the smallest normalized sum of distances within the dataset. SFKM significantly reduces the computation time and shows performance comparable to *PAM*, but it is not effective in improving cluster quality. Also, SFKM tends to perform worse than randomly selecting initial medoids, as reported in (Schubert and Rousseeuw, 2021). SFKM lacks memory optimization as it requires the creation of a distance matrix in memory for the entire dataset.

### 3.2 *k*-medoids Swap Optimization

One of the main challenges of *k*-medoids for large datasets is the high computational cost, leading to long execution times and memory limitations. The *swap* step contributes the most to the computational cost of *PAM*, since it seeks for the best pair of medoid and non-medoid objects, among all possible  $k \times (n - k)$  pairs, which can improve the clustering quality.

Our proposal does not rely on existing methods. Many methods include techniques for parallel clustering algorithms, such as (Vandanov et al., 2023). Statistical estimation techniques have also improved the performance of *k*-medoids, such as BanditPAM (Tiwari et al., 2020), an algorithm inspired by multi-armed bandits. Despite the potential advantages of existing approaches, our proposal concentrates on optimizing the clustering algorithm in its original form without changing the overall problem-solving strat-

egy. Parallel clustering algorithms and statistical estimation techniques have proven beneficial in overcoming computational challenges. However, they often introduce complexities or modifications to the methods. Our goal is to enhance the efficiency and performance of the clustering algorithm while preserving its intrinsic structure and characteristics.

Additionally, other techniques have been introduced to enhance existing approaches, such as faster  $k$ -medoids (Schubert and Rousseeuw, 2021). The work proposes *FastPAMI* and *FasterPAM* algorithms, both bringing an enhancement to the *PAM* algorithm. *FastPAMI* ensures the same output as *PAM*, while *FasterPAM* does not, although maintains equivalent quality. Both approaches improve the  $k$ -medoids complexity in a factor of  $O(k)$ . However, when the number of objects in the dataset  $n$  is much larger than the number of medoids  $k$ , the improvement may not be advantageous, as also observed by (Tiwari et al., 2020). Based on experimental results, *FasterPAM* is faster than *FastPAMI*. Both approaches lack memory optimization, as require a distance matrix as an input, and lack space pruning optimization.

### 3.3 Open Issues

There is a gap in the existing literature concerning the  $k$ -medoids strategy to employ MAMs in main memory. Such a strategy should not require a distance matrix to enhance the *swap* step of the algorithm, which would be beneficial for large datasets. Our proposed *KluSIM* method fills this gap. The baseline competitor is *FasterPAM*, which has a specific focus on improving the *swap* step. In contrast, other related studies have mainly focused on developing new strategies for selecting initial medoids.

We propose a method to improve the *swap* of *PAM*, incorporating a space pruning optimization approach. We aim to optimize memory usage enabling working with large datasets, exchanging the distance matrix with a much smaller distance indexing MAM. We detail our proposal in the next section.

## 4 THE *KluSIM* METHOD

In this work, we propose the *k-medoids clustering Swap Improvement with Metric Access Method (KluSIM)* method, designed to enhance the efficiency of the *swap* step of  $k$ -medoids clustering. *KluSIM* clusters objects in a vector space, satisfies the metric space constraints (see Section 2.2), and takes advantage of MAMs for space pruning.

Algorithm 1 presents the pseudocode of *KluSIM*.

It takes as input: the set of objects; the number of clusters; the number of nearest neighbors of medoid candidates during the *swap* step; and the initialization method. The outputs are the set of  $k$  medoids, and the produced clusters.

Roman numerals in parentheses indicate the steps of the algorithm. In line 2, the algorithm starts instantiating a MAM with the objects in  $S$ , and then selects the initial set of  $k$  medoids (line 3). Line 4 performs the *ASq* operation, which computes overall cluster quality  $TD$  and the current cluster assignment  $\mathcal{C}$  using the initial set of medoids  $M$  (line 4). Then, *KluSIM* repeats the following steps for each cluster (lines 6-17): Computes cluster centroids (line 7); Performs  $k$ -NN search (line 8); Executes *ASq* operation for each object  $o_j$  within the set  $S_p$  of nearest objects of the centroid (line 12); Measures cluster quality improvement (line 13). The loop repeats while  $TD$  decreases, *i.e.* the cluster quality improves.

Algorithm 1: *KluSIM* ( $S, k, p, \text{initMethod}$ ).

---

```

input :  $S$ : Set of objects
input :  $k$ : Number of clusters
input :  $p$ : Number of nearest neighbors of
        medoid candidate
input :  $\text{initMethod}$ : Initialization method (e.g.,
        build, k-means++)
output :  $M$ : Set of  $k$  medoids
output :  $\mathcal{C}$ : Set of clusters

1 begin
2    $\text{mam} \leftarrow \text{CreateMAM}(S)$ ; (i)
3    $M \leftarrow \text{InitializeMedoids}(k, \text{initMethod})$ ; (ii)
4    $TD, \mathcal{C} \leftarrow \text{mam.ASq}(M)$ ; (iii)
5   while  $TD$  decreases do
6     for cluster  $C_i \in \mathcal{C}$  do
7        $\mu_i \leftarrow \text{ComputeCentroid}(C_i)$ ; (iv)
8        $S_p \leftarrow \text{mam.kNNq}(\mu_i, p)$ ; (v)
9       for  $o_j \in S_p$  do
10        /* Check if object  $o_j$  improves cluster quality */
11         $M' \leftarrow M$ ;
12         $m'_i \leftarrow o_j$ ;  $m'_i \in M'$ 
13         $TD', \mathcal{C}' \leftarrow \text{mam.ASq}(M')$ ; (iii)
14        if  $TD' < TD$  then (vi)
15           $TD \leftarrow TD'$ ;
16           $\mathcal{C} \leftarrow \mathcal{C}'$ ;
17          swap  $m_i$  with  $o_j$ ;

```

---

Figure 1 illustrates the workflow of *KluSIM*. It divides the algorithm into six steps, as detailed next. The step numbering is the same as indicated in Algorithm 1.

The initial step (i) of *KluSIM* instantiates a MAM with all objects in the dataset. In step (ii), the algorithm initializes by selecting  $k$  initial medoids using

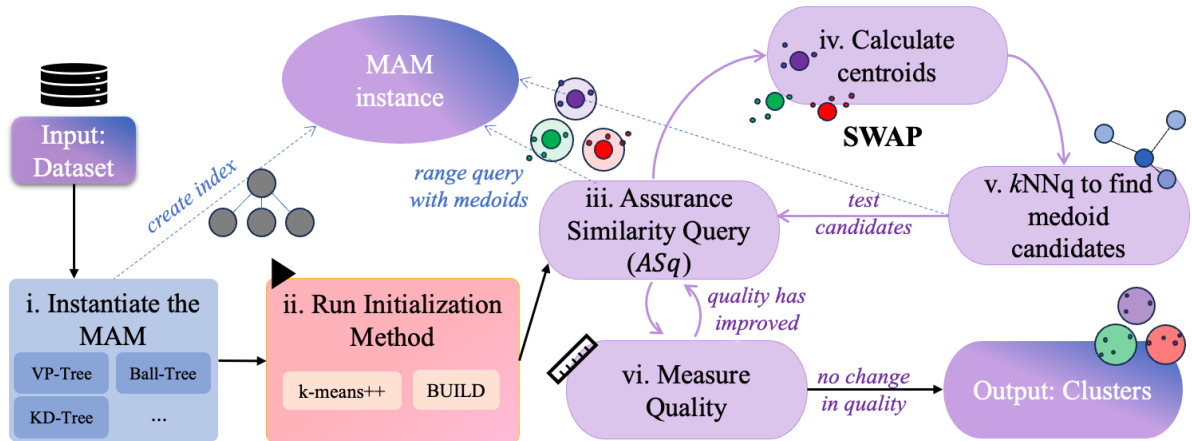


Figure 1: The *KluSIM* algorithm. **i.** Create a MAM with input objects. **ii.** Select  $k$  initial medoids using a given initialization method. **iii.** Perform the Assurance Similarity Query (*ASq*). **iv.** Calculate the centroid for each cluster. **v.** Find the  $p$  nearest objects from each centroid. **vi.** Calculate  $TD$ . Repeat the process while  $TD$  decreases.

an initialization method such as *build* or *k-means++*. Step (iii) executes a search operation, referred to as *Assurance Similarity Query (ASq)*, using the MAM instance. Then, following steps are executed iteratively until convergence:

- **Compute Cluster Centroids (step iv):** For each cluster, compute the centroid object.
- **Perform a  $k$ -NN Search to Find Medoid Candidates (step v):** Based on the centroid of each cluster, select the  $p$  nearest objects to the centroid employing a MAM. In this step, parameter  $p$  is the *number of nearest neighbors*. Taking advantage of MAMs data organization allows to efficiently prune the search space when performing query operations. During the *kNNq*, the MAM discards regions, as they are unlikely to contain nearest neighbors, avoiding unnecessary distance calculations.
- **Execute the assurance similarity query operation *ASq* (step iii):** Analyze each object  $o_j \in S_p$  returned from the *kNNq* as a potential new medoid within a cluster  $C_i \in \mathcal{C}$  (line 9). For each candidate, we replace the current medoid  $m_i$  to  $o_j$  (lines 10-11). Function *ASq* returns the clustering quality  $TD$  and  $k$  clusters (line 12).
- **Measure Cluster Quality Improvement (step 6):** If the non-medoid object  $o_j$  decreases  $TD$ , then execute a *swap* operation between  $m_i$  and  $o_j$ .

In this work, we evaluate our algorithm using the *VP-Tree* MAM. However, other MAMs that rely on using representative objects to partition the data space can be employed.

## 4.1 Assurance Similarity Query (*ASq*)

This operation is designed to easy creating  $k$  groups, based on medoid objects. Function *ASq* gets as input the set of medoids. It assures that each object is assigned to the cluster with the most similar medoid. The operation involves three steps: (1) Definition of coverage radius; (2) Assignment of nearest objects; (3) Association of remaining objects.

### 4.1.1 Definition of Coverage Radius

For each medoid  $m_i \in M$ ,  $|M| = k$ , calculate the distance  $\delta(m_i, m_j)$ , where  $1 \leq j \leq k$  and  $i \neq j$ . The coverage radius  $\xi_i$  for medoid  $m_i$  is defined as  $\delta_{min}/2$ , where  $\delta_{min}$  represents the smallest distance obtained from the distance of  $m_i$  to all other medoids.

Figure 2 illustrates the process to define the coverage radius  $\xi$  of the medoid  $m_1$ . Considering  $k = 3$  (a), the first step computes the distance (b) from the medoid  $m_1$  to  $m_2$  and  $m_3$ . Then, in (c) the coverage radius  $\xi$  of  $m_1$  is determined as  $\delta_{min}/2$ . This process repeats for the remaining medoids in (a).

### 4.1.2 Assignment of Nearest Objects

Figure 3(a) shows the coverage radius of each medoid. For each medoid  $m_i \in M$ , a range query using MAM is conducted with the coverage radius  $\xi_i$ , as in (b). The objects covered by  $\xi_i$  from medoid  $m_i$  will form an initial cluster. Then, *KluSIM* employs the instantiated MAM to prune the search space and efficiently identify regions within the specified radius.

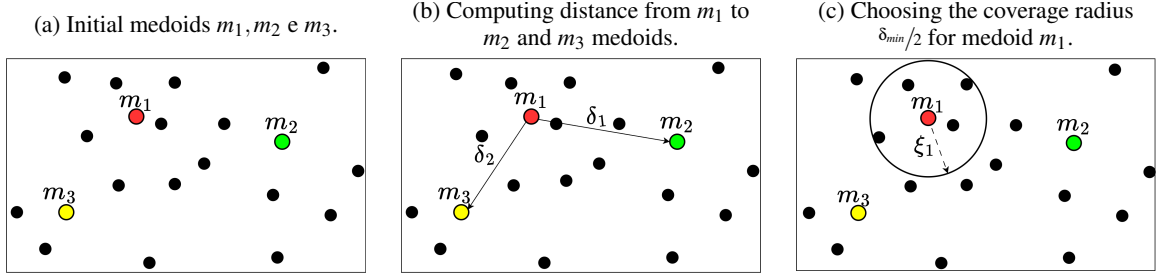


Figure 2: Finding the coverage radius  $\xi$  of medoid  $m_1$ , for  $k = 3$ .

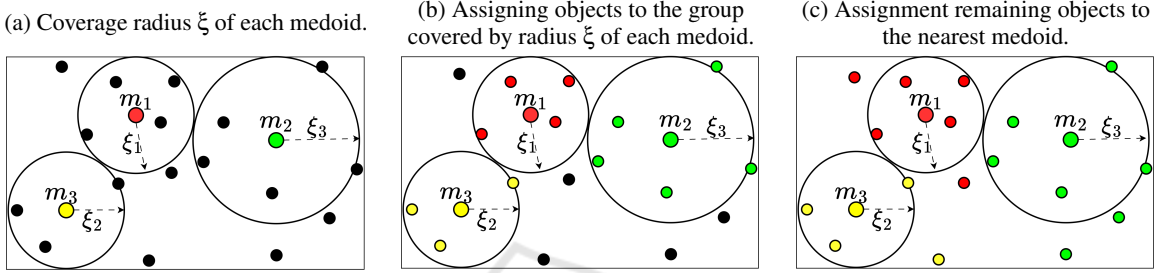


Figure 3: Assigning each object to the nearest medoid, for  $k = 3$ .

#### 4.1.3 Association of Remaining Objects

Objects outside the covering radius from any medoid are treated as exceptions. In Figure 3(c), the algorithm calculates the distance from each non-covered object to all medoids. Then, each object is assigned to the group of the nearest medoid.

### 4.2 Aspects and Advantages of KluSIM

*KluSIM* employs a clever heuristic that relies on pruning the search space to significantly reduce the distance calculations. The *swap* step considers only a subset of objects near the cluster centroid to be considered as better medoids during clustering iterations. The *swap* step is the bottleneck of  $k$ -medoid-based approaches, and *KluSIM* reduces the computational cost of such approaches. The proposed approach benefits clustering tasks over large datasets, since it exchanges the distance matrix of the objects with locally based subsets of distances maintained by MAM. Following, we experimentally evaluate the improvements obtained by *KluSIM*.

## 5 EXPERIMENTS

In this section, we evaluate the efficiency (execution time and number of distance calculations) and effectiveness (cluster quality) of the *KluSIM*.

### 5.1 Datasets and Setup

The experiments were conducted on several datasets:

- **Synthetic Datasets:** All synthetic datasets are from the *make\_blobs* generator (Pedregosa et al., 2011). A total of 48 datasets were created, with dimensions (8, 64, and 128), number of samples (5,000, 25,000, 50,000, and 100,000), and numbers of clusters  $k$  (5, 20, 50, and 100).
- **Real Datasets:** We selected the image datasets *ds-Mammoset* (3457 tuples, 2 clusters) (Oliveira et al., 2017), *ds-DeepLesion* (33334 tuples, 5 clusters) (Yan et al., 2017) and *ds-MNIST* (70000 tuples, 10 clusters) (Lecun et al., 1998). We employed the feature vectors provided by (Cazzolato et al., 2022), generated with descriptors Texture Spectrum (TS, 8 dimensions), Normalized Color Histogram (NCH64, 64 dimensions), and Color Structure (CS, 128 dimensions).

The choice of an optimal initialization method is crucial. We evaluate two initialization strategies: the *build* method and *k-means++*. The results reported for each dataset are the average performance across 10 iterations. In all experiments, our proposed algorithm was compared against the state-of-the-art *FasterPAM* algorithm, our baseline.

All experiments were performed on an Intel® Core™ i5-7300U (2.60Ghz) with 16GB of RAM and Ubuntu 20.04 LTS (64-Bit) GNU/ Linux OS. All algorithms are implemented in Cython with the same

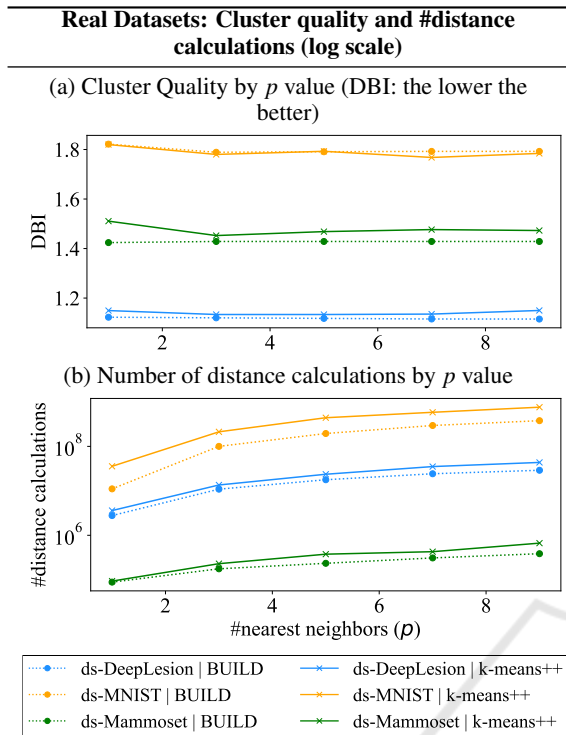


Figure 4: The plots show the cluster quality and number of distance calculations varying  $p$  value. For  $p \geq 3$ , the cluster quality does not improve significantly, but the number of distance calculation increases.

configuration, and none of them utilizes a distance matrix in memory, ensuring a fair comparison.

**Reproducibility.** Our codes are available at <https://github.com/teixeiralari/KluSIM>.

## 5.2 Estimating the Best Value for $p$

In step ( $v$ ) ( $k$ -NN to find medoids candidates) of *KluSIM* (see Figure 1), we perform a  $kNNq$  operation to find the  $p$  objects nearest to the centroid. We evaluate the influence of  $p$  in the cluster quality. We tested *KluSIM* with  $p$  values set to 1, 3, 5, 7, and 9, analyzing the effect on both cluster quality and number of distance calculations. Figure 4 shows (a) the cluster quality and (b) the number of distance calculations across different  $k$  values for real datasets. No significant improvement occurred in the cluster quality for the number of nearest neighbors  $p \geq 3$ , although there is an almost linear increase in the number of distance calculations. Thus, the remaining experiments were conducted using  $p = 3$ .

## 5.3 Evaluating the Execution Time

The efficiency of clustering algorithms is critical for large-scale datasets. We evaluate execution time for *KluSIM* against *FasterPAM*. The primary focus here is to understand how the execution time varies regarding the number of samples and clusters.

Figure 5 shows the experiments over synthetic datasets with the number of samples from 5,000 to 100,000, and the average run time with *build* and *k-means++* initialization. All results are averaged across each dimension (8, 64, 128), for each dataset. We compared the results of *KluSIM* and *FasterPAM*. In (a), the scenario with a small number of clusters ( $k = 5$ ) and a large number of samples ( $S = 100,000$ ) using *build* initialization, *FasterPAM*'s *swap* run time was 5,144 seconds, and *KluSIM* took 5.96 seconds, being approximately 863 times faster. With *k-means++* initialization, *KluSIM* outperformed *FasterPAM* by a speedup factor of 881 times. As the number of clusters increased to  $k = 100$  with large sample size of  $S = 100,000$  using *build* (Figure 5-d), a speedup of about 35 times was observed. Similarly, with *k-means++*, *KluSIM* exhibited still a substantial speedup of about 16 times compared to *FasterPAM*.

Figure 6 shows the results for real datasets. Specifically, for *ds-MammoSet* with a dimension of 8 (a), there is an approximate 21-fold acceleration in the *swap* step with the *build* initialization and roughly 35 times using *k-means++* when compared to *FasterPAM* under the same conditions. On the other hand, for *ds-MNIST* dataset, our algorithm achieved a speedup of approximately 99 times with *build* initialization and 123 times with *k-means++* in comparison to *FasterPAM* in the same scenarios. Furthermore, considering a higher dimension (c), the gain of our method is nearly 40 times using *build* or *k-means++* initialization for *ds-MammoSet*, and 275 times for *ds-MNIST* dataset using *build* initialization.

## 5.4 Evaluating the Number of Distance Calculations

We evaluated the number of distance calculations required in the clustering process for both *KluSIM* and *FasterPAM*, varying the number of samples and clusters. This analysis aims to understand the computational cost associated with *KluSIM* when compared with the baseline *FasterPAM* approach. Given that one of the goals of *KluSIM* is to provide efficiency and scalability when clustering large datasets, as well as reducing potential computational overhead in environments with limited memory, we conducted the experiments, for both *KluSIM* and *FasterPAM*, with-



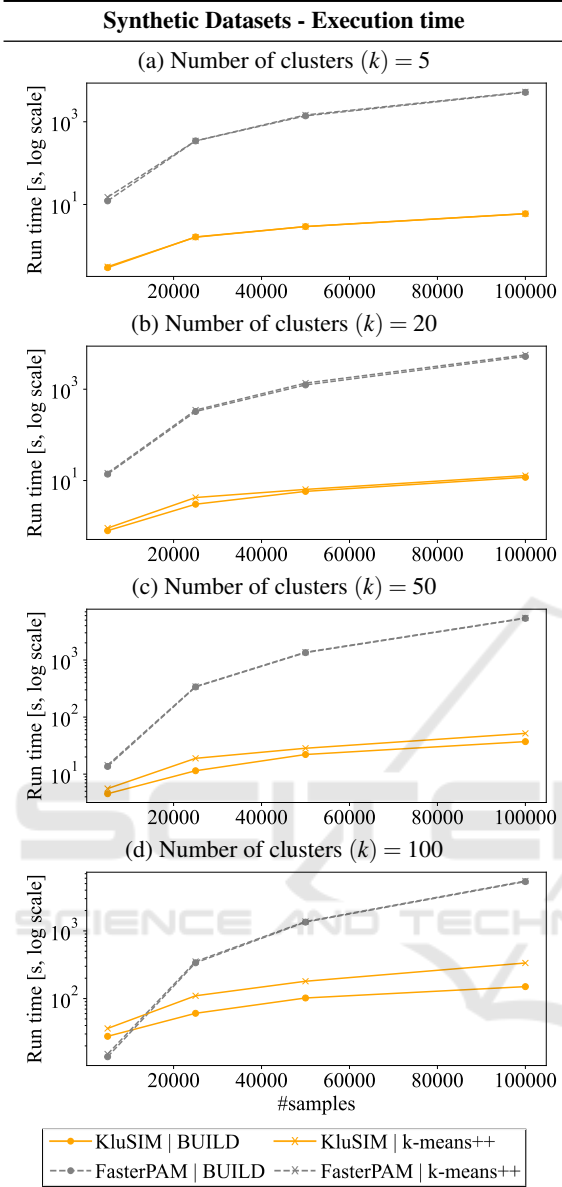


Figure 5: *KluSIM* outperforms the main competitor by up to 881 times on synthetic datasets of different sizes. In nearly all cases, *KluSIM* achieves faster times.

out storing a distance matrix in memory. All required distances were calculated on the fly.

Figure 7 displays the total number of distance calculations taken by the algorithms in the clustering process, for both *build* and *k-means++* initialization. The results are averaged across all dimensions (8, 64, 128) for each dataset. With a large number of samples ( $S = 100,000$ ) and a small number of clusters ( $k = 5$ ) (a), our proposed algorithm *KluSIM* converges with approximately 3,500 times fewer distance calculations than *FasterPAM*, using *build* or *k-means++*.

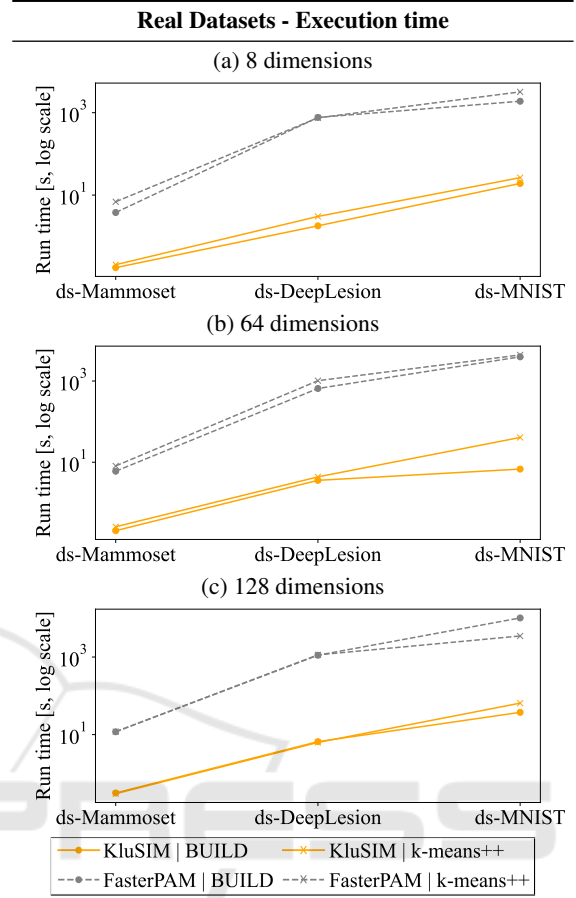


Figure 6: *KluSIM* is consistently faster than the main competitor for real datasets. The plots show the run time of *KluSIM* and *FasterPAM* on real datasets. In all cases, *KluSIM* speeds up clustering.

As the number of clusters increases to  $k = 100$  (d), there is a notable reduction of roughly 39 times for *build* initialization and 15 times for *k-means++* initialization in the number of distance calculations in comparison with the baseline *FasterPAM*.

Figure 8 presents the results for real datasets. With 8 dimensions (a), our method exhibits a significant reduction in the number of distance calculations. Specifically, there is a 184-fold reduction when opting for the *build* initialization, compared to a 151-fold reduction with *k-means++* initialization for a smaller dataset (*ds-Mammaset*). In contrast, for a larger dataset (*ds-MNIST*), there is a reduction of 59 times in the number of distance calculations with *build*, and 56 times with *k-means++*. All these improvements are observed in comparison to *FasterPAM* under same conditions. Considering a highest dimensionality (c), *KluSIM* reduces by 62 times the number of distance calculations with *build* initialization, and 54 times with *k-means++* initialization for the *ds-*

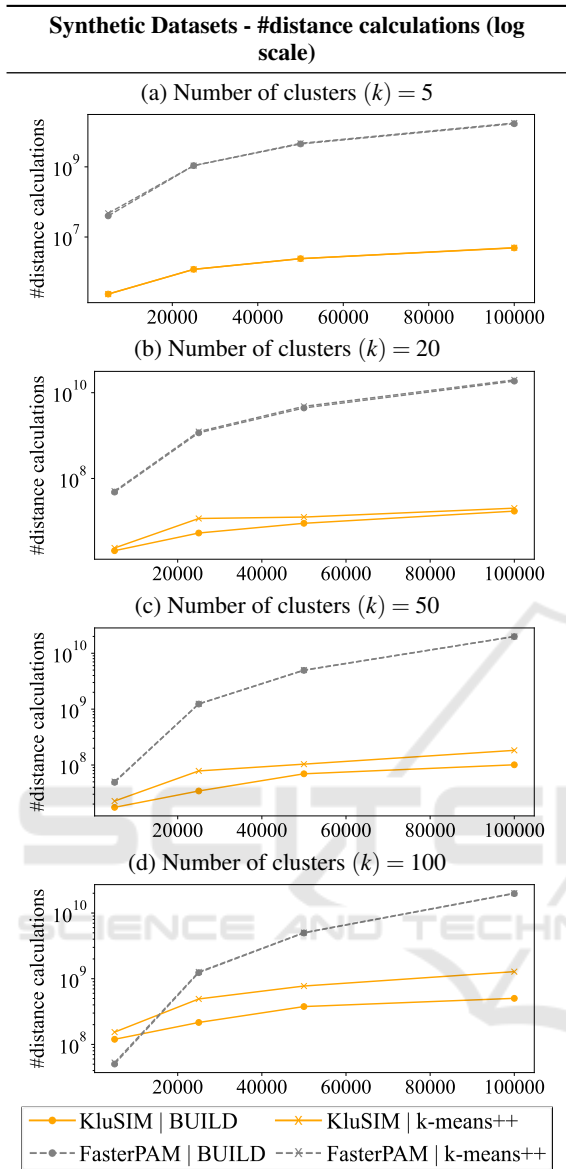


Figure 7: *KluSIM* reduces the number of distance calculations by up to 3,500 times when compared to *FasterPAM*. The plots show the number of distance calculations on synthetic datasets for *KluSIM* and *FasterPAM*.

*Mammaset*. For the *ds-MNIST* dataset, *KluSIM* needs 193 times fewer distance calculations with *build* initialization when compared with *FasterPAM*.

### 5.5 Evaluating Cluster Quality

Finally, we evaluated the cluster quality. Enhancing the speed of clustering algorithms is essential for handling large datasets, while maintaining or even improving the quality of the results. We investigate how our proposal *KluSIM* strikes a balance between time

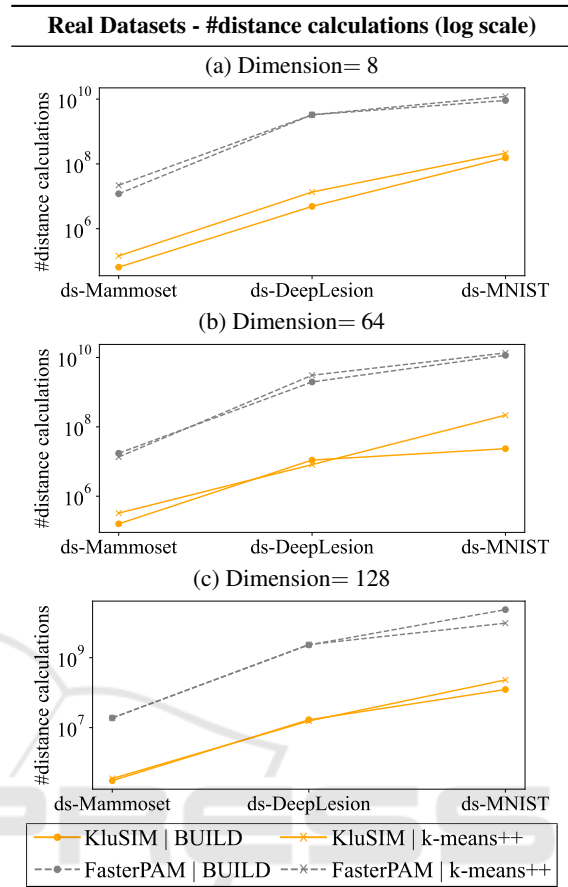


Figure 8: *KluSIM* shows a reduction in the number of distance calculations in the clustering process when compared to *FasterPAM*. The plots show the number of distance calculations on real datasets for *KluSIM* and *FasterPAM*.

efficiency and cluster quality in comparison to *FasterPAM* employing synthetic and real datasets.

Figure 9 displays the average silhouette score (*SS*) across all dimensions categorized by the number of clusters and samples over synthetic datasets. The results show that *KluSIM* exhibits comparable cluster quality to *FasterPAM*. Figure 10 illustrates the cluster quality over real datasets. Similar to the synthetic datasets, we observed an equivalent quality between our method *KluSIM* and *FasterPAM*.

### 5.6 Discussion

The choice of the initialization method is a crucial aspect that influences our method *KluSIM* convergence and overall cluster quality. Our experiments show the *build* initialization method outperforming *k-means++* in terms of quality, time efficiency, and number of distance calculations required for convergence. This superiority appears because *build* chooses each medoid

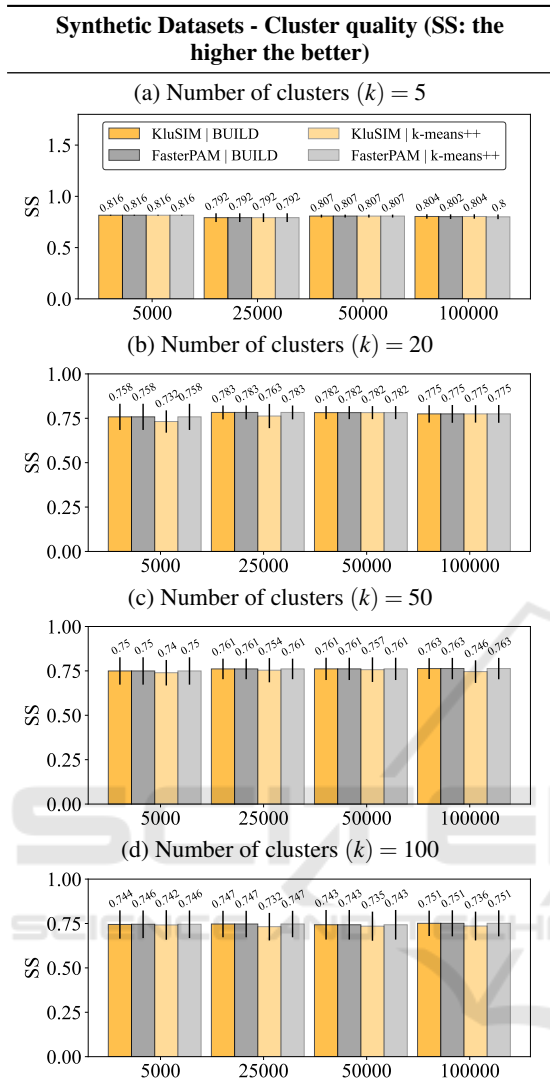


Figure 9: *KluSIM* quality of clustering is equivalent to the quality of main competitor. Cluster quality over synthetic datasets.

with an optimal quality (*i.e.* less *TD*). On the other hand, *k-means++* randomly selects the first and subsequent objects that are likely to be far from each other, without a focus on minimizing *TD*. Notably, using *build* showcased a substantial speed-up of up to 881 times. The improvement was particularly high in scenarios where the number of clusters  $k$  was substantially smaller than the number of objects in the dataset  $S$  ( $k \ll n$ ).

All in all, *KluSIM* has shown a significant reduction in the number of distance calculations, speeding up existing approaches, and maintaining comparable quality results.

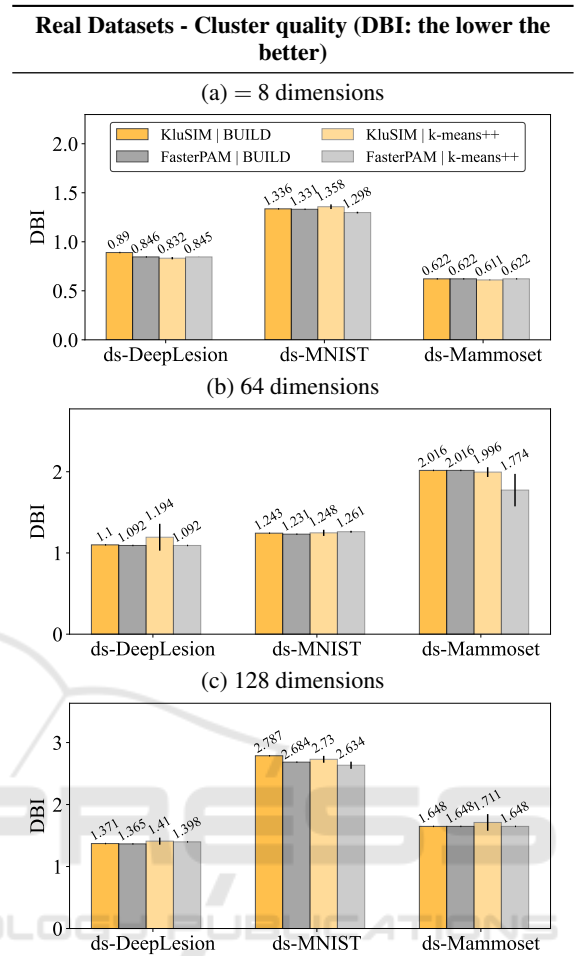


Figure 10: *KluSIM* quality of clustering is equivalent to the quality of the main competitor. Cluster quality over real datasets.

## 6 CONCLUSION

This paper introduced the *KluSIM* approach, designed to improve the efficiency of *k*-medoids algorithms by incorporating a metric access methods to speed up clustering. *KluSIM* focuses on the *swap* step process, exhibiting significant improvements. Through experimentation, *KluSIM* was compared to the baseline *FasterPAM*, significantly reducing time in clustering processing.

Our algorithm exhibited an average speedup of up to 881 times when compared with the baseline *FasterPAM*, and a reduction of up to 3,500 times in distance calculations, whereas maintaining a comparable clustering quality. Unlike several methods in the literature, our algorithm does not store a distance matrix in memory, eliminating a potential bottleneck faced by other methods when clustering large datasets.

Our *KluSIM* proposal stands out as an efficient and scalable solution for *k*-medoids clustering tasks. The combination of metric access methods, optimized initialization heuristics, and the elimination of the need for a distance matrix in memory collectively contribute to the outstanding performance gains. Thus, *KluSIM* is a powerful tool for scalable and high-performance clustering tasks, particularly in scenarios with limited computational resources or large datasets.

In future work, we intend to explore MAM-based initialization heuristics to leverage the index structure in the entire clustering process. We also want to evaluate *KluSIM* with other distance functions.

## ACKNOWLEDGEMENT

This research was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and 12620352/M, by the São Paulo Research Foundation (FAPESP, grants 2016/17078-0, 2020/11258-2), the National Council for Scientific and Technological Development (CNPq) and JIT Educação.

## REFERENCES

- Arthur, D. and Vassilvitskii, S. (2007). K-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035.
- Barioni, M. C. N., Razente, H. L., Traina, A. J., and Traina Jr, C. (2008). Accelerating k-medoid-based algorithms through metric access methods. *Journal of Systems and Software*, 81(3):343–355.
- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.
- Cazzolato, M. T., Scabora, L. C., Zabet, G. F., Gutierrez, M. A., Traina Jr, C., and Traina, A. J. (2022). Featset+: Visual features extracted from public image datasets. *Journal of Information and Data Management*, 13(1).
- Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.
- Han, J., Kamber, M., and Pei, J. (2011). *Data Mining: Concepts and Techniques*, 3rd edition. Morgan Kaufmann.
- Kaufman, L. (1990). Rousseeuw, pj: Finding groups in data: An introduction to cluster analysis. *Applied Probability and Statistics, New York, Wiley Series in Probability and Mathematical Statistics*.
- Kenger, O. N., Kenger, Z. D., Özceylan, E., and Mrugalska, B. (2023). Clustering of cities based on their smart performances: A comparative approach of fuzzy c-means, k-means, and k-medoids. *IEEE Access*, 11:134446–134459.
- Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lessen, L. and Schubert, E. (2024). Medoid silhouette clustering with automatic cluster number selection. *Information Systems*, 120:102290.
- Oliveira, P. H., Scabora, L. C., Cazzolato, M. T., Bedo, M. V., Traina, A. J., and Traina-Jr, C. (2017). Mammot: An enhanced dataset of mammograms. In *Satellite Events of the Brazilian Symp. on Databases. SBC*, pages 256–266.
- Omohundro, S. M. (1989). *Five balltree construction algorithms*. International Computer Science Institute Berkeley.
- Park, H.-S. and Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.
- Qaddoura, R., Faris, H., and Aljarah, I. (2020). An efficient clustering algorithm based on the k-nearest neighbors with an indexing ratio. *International Journal of Machine Learning and Cybernetics*, 11(3):675–714.
- Ran, X., Xi, Y., Lu, Y., Wang, X., and Lu, Z. (2023). Comprehensive survey on hierarchical clustering algorithms and the recent developments. *Artificial Intelligence Review*, 56(8):8219–8264.
- Schubert, E. and Rousseeuw, P. J. (2021). Fast and eager k-medoids clustering: O(k) runtime improvement of the pam, clara, and clarans algorithms. *Information Systems*, 101:101804.
- Tiwari, M., Zhang, M. J., Mayclin, J., Thrun, S., Piech, C., and Shomorony, I. (2020). Banditpam: Almost linear time k-medoids clustering via multi-armed bandits. In *NeurIPS*.
- Traina, C., Traina, A., Faloutsos, C., and Seeger, B. (2002). Fast indexing and visualization of metric data sets using slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):244–260.
- Vandanov, S., Plyasunov, A., and Ushakov, A. (2023). Parallel clustering algorithm for the k-medoids problem in high-dimensional space for large-scale datasets. In *2023 19th International Asian School-Seminar on Optimization Problems of Complex Systems (OPCS)*, pages 119–124.
- Yan, K., Wang, X., Lu, L., and Summers, R. M. (2017). Deeplesion: Automated deep mining, categorization and detection of significant radiology image findings using large-scale clinical lesion annotations. *arXiv preprint arXiv:1710.01766*.
- Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, volume 66, page 311. SIAM.
- Zeuzala, P., Amato, G., Dohnal, V., and Batko, M. (2006). *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media.